Opinnäytetyö (AMK)

Tietotekniikan koulutusohjelma

Sulautetut järjestelmät

2014

Tero Elmroos

SSSP-JÄRJESTELMÄN INTEGROINTI DIGITAL SIGNAGE -JÄRJESTELMÄÄN

TURUN AMMATTIKORKEAKOULU
TURKU UNIVERSITY OF APPLIED SCIENCES

OPINNÄYTETYÖ (AMK) | TIIVISTELMÄ

TURUN AMMATTIKORKEAKOULU

Tietotekniikan koulutusohjelma | Sulautetut järjestelmät

Syksy 2014 | Sivumäärä: 31

Lehtori Tiina Ferm

Tero Elmroos

SSSP-JÄRJESTELMÄN INTEGROINTI DIGITAL SIGNAGE -JÄRJESTELMÄÄN

Tässä opinnäytetyössä kuvattiin Samsung Smart Signage Platform -järjestelmässä toimivan sovelluksen yhdistäminen Qem Software Oy:n Digital Signage -järjestelmään. Työssä paneuduttiin sovelluksen ohjelmakoodiin, jolla yhdistettiin sovellus ja Digital Signage -järjestelmä.

Sovellus toteutettiin JavaScript-ohjelmointikielellä hyödyntäen jQuery-kirjastoa ja sen ajax-pyyntöjä. XML-tiedostojen datan hakeminen toteutettiin ajax-pyynnöillä. Datan jäsentäminen suoritettiin jQuery-kirjaston metodeilla. Samsung Smart Signage Platform -järjestelmä yhdistettiin palvelimelle HTTP-osoitteen avulla.

Työn tavoitteena oli yhdistää sovellus ja Digital Signage -järjestelmä toimivaksi kokonaisuudeksi luomalla sovellukseen yhdistämisen tekevä ohjelmakoodi. Tavoitteena työn ohjelmakoodille oli tehdä siitä mahdollisimman helppolukuinen ja rakenteeltaan looginen. Työn tavoitteeseen päästiin hyvin ja tuloksena saatiin toimiva kokonaisuus. Ohjelmakoodista tuli lyhyt, selvä ja rakenteeltaan looginen. Tulokset mahdollistivat Samsung älytelevision käytön Qem Digital Signage -järjestelmän kanssa.

ASIASANAT:

JavaScript, XML, Ajax, Integrointi, Jäsentäminen, Smart Signage Platform

BACHELOR´S THESIS | ABSTRACT

TURKU UNIVERSITY OF APPLIED SCIENCES

Degree programme in Information Technology | Embedded Software

Autumn of 2014 | Total number of pages: 31

Tiina Ferm, M.Sc., Lecturer

Tero Elmroos

INTEGRATION OF SSSP-SYSTEM TO DIGITAL SIGNAGE-SYSTEM

The purpose of this study was to describe the integration of the Samsung Smart Signage Platform system and the Qem Software's Digital Signage system. The main objective of this study was to explain the code behind the application which combined the two systems.

The application was created with the JavaScript-programing language and it used the jQuery library and its Ajax requests. The fetching of data in XML-files was created with Ajax requests. The parsing of the XML data was accomplished with the jQuery library. An HTTP connection was used to connect the Samsung Smart Signage Platform system to the server.

The main goal of this study was to integrate the application and the Digital Signage system to a functional application. The goal for the programing code was to make it logically structured and easy to read. The goal of the study was achieved and resulted in a functional application. The goal for the programming code was also achieved. The results of the study has enabled the use of the Samsung smart TV with the Qem Digital Signage system.

KEYWORDS:

JavaScript, XML, Ajax, Integration, Parsing, Smart Signage Platform

SISÄLTÖ

KÄYTETYT LYHENTEET **6**

1 JOHDANTO **7**

2 TEORIA **8**

2.1 XML-merkintäkieli 8

2.1.1 Well-formed XML 8

2.1.2 XML-skeema 9

2.2 jQuery-kirjasto 9

2.3 Ajax 10

2.4 Digital Signage 10

2.5 Samsung Smart Signage Platform 11

3 KÄYTÄNNÖN TOTEUTUS **12**

3.1 Työvälineet 13

3.1.1 Ohjelmointialusta 13

3.1.2 Ohjelmointikieli 13

3.1.3 XML-skeema 13

3.1.4 Samsung Smart Signage Platform 14

3.2 Ohjelmakoodin toteutus 14

3.2.1 Muuttujat 14

3.2.2 Ajax-pyynnöt 16

3.2.3 XML-tiedoston datan jäsentely 18

3.2.4 Muuttuneiden listan muodostus 19

3.2.5 Latauslistan muodostus 20

3.2.6 Soittolistojen muodostus 21

3.2.7 Testaus ja virheiden etsiminen 24

4 YHTEENVETO **26**

LÄHTEET **27**

LIITTEET

Liite 1. Ohjelmakoodi XML-tiedoston datan jäsentämiseen.
Liite 2. Ohjelmakoodi muuttuneiden listan muodostamiseen.

KUVIOT

Kuvio 1. Tilakaavio järjestelmien integroimisen metodeista. 12

OHJELMAKOODIT

Ohjelmakoodi 1. Muuttujat. 14
Ohjelmakoodi 2. Parser.GETLocalXML-metodi. 16
Ohjelmakoodi 3. Parser.GETExternalXML-metodi. 17
Ohjelmakoodi 4. Parser.createDownloadList-metodi. 20
Ohjelmakoodi 5. Parser.createPlaylist-metodi. 22
Ohjelmakoodi 6. Parser.createNewPlaylist-metodi. 23
Ohjelmakoodi 7. Main.log-metodi. 24

KÄYTETYT LYHENTEET

Ajax	Asynchronous JavaScript And XML, joukko verkko-ohjelmointitekniikoita
HTML	HyperText Markup Language, kuvauskieli, jolla näytetään informaatiota
HTTP	HyperText Transfer Protocol, tiedonsiirtoprotokolla, joka on käytössä verkkoselaimilla ja WWW-palvelimilla
jQuery	Avoimen lähdekoodin JavaScript-kirjasto, joka on tarkoitettu kaikilla selaimilla toimivaksi
MD5	Message-digest -algoritmi, joka tuottaa viestistä 128-bittisen tiivisteen
SSSP	Samsung Smart Signage Platform, Samsungin älytelevision käyttämä järjestelmä
XML	eXtensible Markup Language, merkintäkieli, jolla kuvataan informaatiota

1 JOHDANTO

Qem Software Oy:n Qem Digital Signage -järjestelmän yhdistämiseen asiakaskohteessa olevaan näyttölaiteeseen käytetään mediasoittimeksi soveltuvaa tietokonetta. Tietokonetta käytetään, koska normaalin näyttölaiteen kautta ei ole mahdollista yhdistää palvelimelle tai jäsentää dataa näytettäväksi näyttölaiteen ruudulla. Samsung Smart Signage Platform -järjestelmä mahdollistaa palvelimelle yhdistämisen sekä datan jäsentämisen älytelevision avulla.

Opinnäytetyössä kuvataan Samsung Smart Signage Platform -järjestelmän yhdistäminen Qem Digital Signage -järjestelmään XML-merkintäkielen avulla. Työn toimeksiantajana toimii Qem Software Oy. Opinnäytetyö on osa Samsung Smart Signage Platform -järjestelmässä toimivaa sovellusta. Opinnäytetyön työosuus on toteutettu Qem Software Oy:lle tehdyn sovelluksen rakentamisen yhteydessä vuoden 2014 syksyllä.

Teoriaosuudessa paneudutaan työssä käytettävien työvälineiden ja järjestelmien käyttöön ja teoriaan. Työn toteutusosiossa selvennetään lyhyesti työssä käytettävät työvälineet ja syy minkä takia niitä käytetään. Ohjelmakoodin esimerkkien avulla käsitellään ohjelmakoodin toiminta ja kerrotaan lyhyesti ohjelmakoodin tarkoituksesta ja sen liittymisestä toimeksiantajalle tehtyyn sovellukseen. Työ keskittyy järjestelmien yhdistämiseen ja järjestelmien välillä kulkevan tiedon jäsentämiseen.

2 TEORIA

Tässä luvussa käsitellään järjestelmien liittämiseen käytettyjen työkalujen teoriaa ja selvitetään lyhyesti mitä työssä olevat järjestelmät ovat. Työkaluista selvennetään myös mitä niillä voidaan tehdä ja miksi niitä kannattaa käyttää.

2.1 XML-merkintäkieli

XML on merkintäkieli, jolla kuvataan tiedon rakennetta ilman ennalta määrättyjä elementtejä. XML:ään sijoitetaan tiedon kuvaus tekstimuodossa hyvin yksinkertaisella ja selvällä rakenteella. Tämä tarkoittaa, että XML-tiedosto vain säilyttää tietoa eikä tee mitään muuta. [1]

XML:n käyttö tulee tarpeelliseksi esimerkiksi laiteläheisessä ohjelmoinnissa, jolloin ei ole käytettävissä tai ei ole mahdollista käyttää erillistä tietokantaa tietojen tallentamiseen ja hakemiseen. XML tarjoaa myös yleisesti selvän ja kestävän tavan siirtää ja tallentaa informaatiota, sillä sen avulla tallennettu tieto on tekstimuodossa. [2]

2.1.1 Well-formed XML

Well-formed XML tarkoittaa, että XML-dokumentissa ei ole virheitä eikä dokumentin elementtejä ole määritelty ristiin keskenään. Elementtien ristiin määrittäminen tarkoittaa, että elementin aloitusmerkki sijaitsee yhden elementin sisällä ja lopetusmerkki sijaitsee toisen elementin sisällä. Elementit tulee myös nimetä loogisesti niin, että niiden nimet kuvaavat sisältöä. [3]

XML-dokumentin syntaksin oikeaoppisuuden voi tarkistaa seuraavien sääntöjen avulla:

- XML-dokumentissa tulee olla juurielementti.
- Jokaisella XML-elementillä tulee olla lopetusmerkki.
- XML-elementit ovat merkkikokoriippuvaisia.

- Kaikki XML-elementit pitää olla oikeaoppisesti sisäkkäin.
- Attribuuttien arvot tulee olla lainausmerkeissä.

Kaikkien sääntöjen täyttyessä XML-dokumenttia voidaan sanoa well-formed XML-dokumentiksi. [4]

XML-dokumentin tulee seurata well-formed dokumentin sääntöjä sen datan käsittelyn mahdollistamiseksi. Esimerkiksi mikäli dokumentissa on määritetty elementit ristiin tai XML-elementillä ei ole lopetusmerkkiä, palauttaa sen datan lukemisyritys virheen ja pahimmassa tapauksessa pysäyttää koko sovelluksen suorittamisen. [5]

2.1.2 XML-skeema

XML-skeema on XML-dokumentin rakennekuvaus, jonka avulla määritellään useita asioita XML-dokumentista. Määriteltyjä asioita voivat olla esimerkiksi elementtien nimet, tyypit, esiintymiset ja niiden sisältämät attribuutit. XML-skeeman mukainen dokumentti voidaan luokitella päteväksi ja well-formed dokumentiksi. [6]

XML-skeemaa käytettäessä XML-dokumenttien rakenne pysyy samana, vaikka niiden sisältämä tieto muuttuu. Tämä mahdollistaa esimerkiksi usean XML-dokumentin käyttämisen yhdessä sovelluksessa tai XML-dokumentin päivittämisen ilman sovelluksen päivittämistä. XML-skeeman määrätty rakenne tarjoaa myös standardoidun tavan vaihtaa tietoa alustojen välillä. [6]

2.2 jQuery-kirjasto

jQuery-kirjasto on nopeasti latautuva, kooltaan pieni ja useita ominaisuuksia sisältävä JavaScript-kirjasto. Se on ilmainen, avoimen lähdekoodin, kaikille selaimille tarkoitettu kirjasto, jonka syntaksi on tehty mahdollisimman helposti ymmärrettäväksi. [7]

jQuery-kirjaston avulla voidaan toteuttaa monimutkaisia selaimen toimintoja yksinkertaisilla käskyillä. Sillä voidaan myös nopeuttaa sovelluksen toimintaa ehkäisemällä ylimääräisten ja laajojen koodirakenteiden syntymistä. Kehittäjän ei myöskään tarvitse opetella uutta ohjelmointikielen syntaksia, koska kirjasto on kirjoitettu JavaScript-ohjelmointikielellä. [8]

2.3 Ajax

Ajax on joukko verkko-ohjelmoinnin tekniikoita ja lähestymistapoja. Tämä tarkoittaa kuitenkin, että Ajax ei ole itsessään tekniikka, vaan se on kokoelma useista tekniikoista, joita voidaan käyttää Ajaxin kautta. [9]

Ajaxin avulla voidaan keskustella verkkopalvelimen kanssa ilman koko verkkosivun päivittämistä, koska Ajax vaihtaa pieniä määriä dataa palvelimen kanssa asynkronisesti verkkosivun taustalla. Tämä luo verkkosivusta vuorovaikutteisemman ja mahdollistaa verkkosivuun tehtävien muutosten päivittämisen ilman verkkosivun uudelleen lataamista. [9]

2.4 Digital Signage

Digital Signage on etähallittua digitaalisen sisällön toistamista julkisella paikalla sijaitsevien näyttöjen verkossa. Tarkoituksena sillä on esimerkiksi välittää informaatiota tai mainostaa jotakin asiaa tai tuotetta. Näyttölaitteena voi toimia mikä vain näyttölaite, mikä kykenee toistamaan sille syötettyä dataa. [10]

Digital Signagen vahvuutena on sen sisällön muokaamisen helppous. Digital Signagessa on yleensä taustalla tietokone tai toistolaite, joka kontrolloi näyttölaitteella toistettavaa mediaa paikallisesti tai verkon yli. Sisältönä toistettavassa esityksessä voi olla erityyppisiä informaatiota tai mainontaa sisältäviä medioita, kuten kuvia tai videoita. [10]

Opinnäytetyön toimeksiantajan Qem Digital Signage -järjestelmä on hyvä esimerkki perinteisestä Digital Signage -järjestelmästä. Sen avulla asiakas voi

verkkosivulla määrittää esityksen sisällön. Esitys tallennetaan palvelimelle ja päivitetään verkon yli toistolaitteena toimivalle tietokoneelle, jonka avulla esitys näytetään näyttölaitteessa. [11]

2.5 Samsung Smart Signage Platform

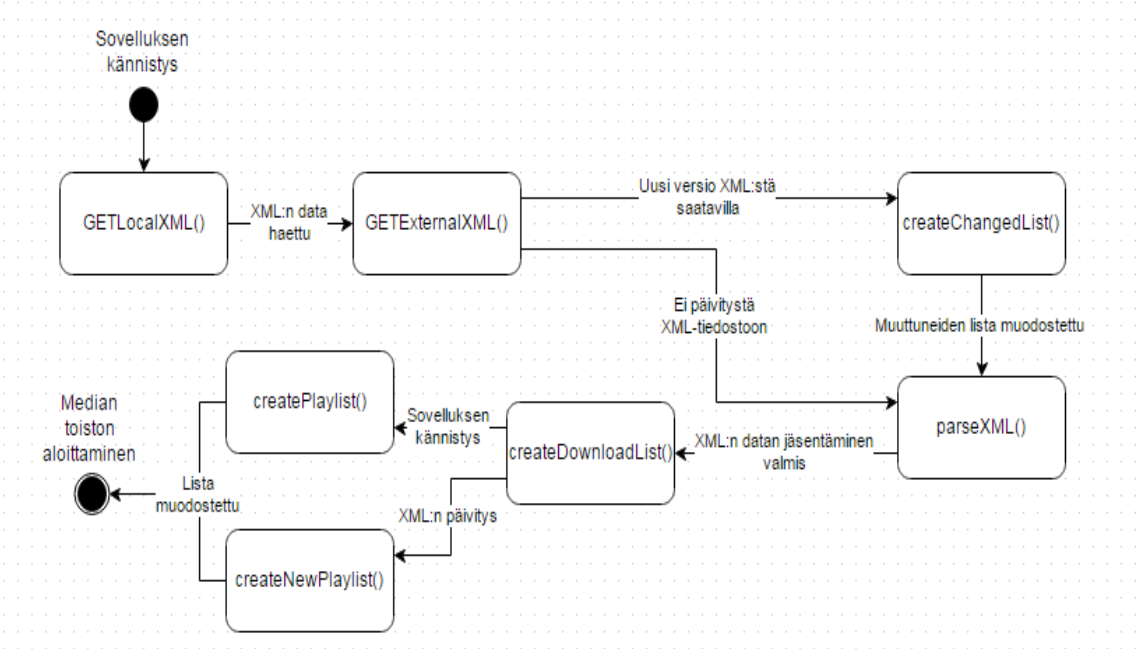
Samsung Smart Signage Platform on Samsungin älytelevisiossa toimiva järjestelmä. Sen tarkoituksena on vähentää kustannuksia ja helpottaa Digital Signage -menetelmien käyttöä, integroimalla älytelevisioon toistolaitteen ja siten poistamalla tarpeen ylimääräisille toistolaitteille. [12]

Smart Signage Platformin omaavan älytelevision asentaminen on pyritty tekemään niin helpoksi, että se mahdollistaa järjestelmän käyttöönoton ilman teknistä henkilöstöä tai tukea. Tällöin älytelevisio voidaan lähettää suoraan asiakkaan määrittämään kohteeseen, jossa asiakas kytkee sen virtalähteeseen ja internet-verkkoon. Verkkoon kytkemisen jälkeen asiakas yhdistää älytelevision palveluntarjoajan antamaan HTTP-osoitteen, josta älytelevisio hakee tarvittavat ohjelmistot ja tiedot esitysten toistamiseen. [13]

3 KÄYTÄNNÖN TOTEUTUS

Tässä luvussa käsitellään työssä käytettävät työvälineet ja kerrotaan työn toteutuksesta. Työn toteutusosassa käsitellään varsinaisen työn toteuttamista ja kuvataan ohjelmakoodin rakenne. Toteutusosiossa viitataan myös sovellukseen, jonka osaksi työ on tehty.

Kuvio 1 havainnollistaa toteutusosassa käsiteltävän ohjelmakoodin suorittamisjärjestystä. Tilakaaviosta on jätetty pois sovelluksen metodit, joilla ei ole vaikutusta järjestelmien integroimiseen.



Kuvio 1. Tilakaavio järjestelmien integroimisen metodeista.

Toteutusosan ohjelmakoodilla on kaksi suorituspolkua. Ensimmäinen polku suoritetaan sovelluksen käynnistyksen yhteydessä, jolloin luodaan ensimmäinen soittolista. Toinen polku suoritetaan soittolistan päivityksen yhteydessä, jolloin luodaan uusi soittolista.

3.1 Työvälineet

3.1.1 Ohjelmointialusta

Ohjelmointialustana käytetään alustariippumatonta Sublime Text 3 tekstieditoria sen ominaisuuksien, kuten useamman tiedoston samanaikaisesti näyttämisen näytöllä, takia. Editorin useat näppäinkomennot sekä mahdollisuus valita ja muokata yhtäaikaisesti monta kohtaa, helpottavat koodin muokkaamista ja rakentamista loogiseksi.

3.1.2 Ohjelmointikieli

XML-tiedoston lukeminen toteutetaan JavaScript-ohjelmointikielellä hyödyntäen jQuery-kirjastoa ja sen ajax-pyyntöjä. JavaScript-ohjelmointikieltä käytetään HTML-tiedoston kautta, koska SSSP-järjestelmä kykenee yhdistämään vain HTTP-osoitteeseen tai lukemaan älytelevision sisäisessä muistissa olevaa HTML-tiedostoa.

3.1.3 XML-skeema

XML-skeemana käytetään toimeksiantajan Digital Signage -järjestelmissä käytettyä skeemaa. Skeemassa määrätään vähintään presentaation alue näyttölaitteella, presentaation sisältö ja presentaatioissa käytettävät tiedostot. XML-skeemassa määritetään myös monia muita elementtejä, joiden käyttö XML-tiedostossa on valinnaista ja tapauskohtaista.

Työssä keskitytään skeemassa oleviin sovellukselle keskeisiin elementteihin, jotka ovat presentaation sisältö ja presentaatiossa käytettävät tiedostot. Presentaation aluetta ei huomioida työssä, koska SSSP-järjestelmä ei tue monen presentaation yhtäaikaista näyttämistä.

3.1.4 Samsung Smart Signage Platform

Työssä käytetään Samsungin DM32D-mallia älytelevisiosta, jossa on T-GFSLDDWWC-1016.5-ohjelmistoversio ja T-GFSLDWWS1-1022-sub-micom-versio. Älytelevisio yhdistetään toimeksiantajan verkkoon RJ-45-liitintyypin omaavan parikaapelin avulla. Verkossa älytelevisio yhdistetään HTTP-yhteydellä, Apachen virtuaalisen lähiverkon kautta tietokoneelle, joka toimii sovelluksen kehitysvaiheessa testipalvelimena.

3.2 Ohjelmakoodin toteutus

3.2.1 Muuttujat

Sovelluksen Ohjelmakoodin 1 kirjoittamisen alussa määritellään ja alustetaan tarvittavat muuttujat. Muuttujiksi määritetään *presentation*-, *contents*-, *content*-, *file*- ja *Parser*-oliot.

Ohjelmakoodi 1. Muuttujat.

```
var presentation = {contentId:[], duration:[]};
var contents = {baseurl:"", version:"", edit:""};
var content = {id:[], contentType:[], version:[]};
var file = {parentId:[], name:[], md5:[]};
var Parser = {
   xmlLocal: null,
   xmlExternal: null,
   newPlaylist: {url:[], type:[], duration:[], items:0}
};
```

Ohjelmakoodin 1 p*resentation*-olion ominaisuuksiin kuuluu *contentID*-jono, jonka alkiot määrittävät presentaatioon kuuluvat XML-tiedostossa olevat content-tiedostot, sekä *duration*-jono, johon alkioiksi kerätään presentaation kestot. Riippuen presentaatiossa käytettävien medioiden tyypeistä, kesto voi

olla myös tyhjä, jolloin jonoon lisätään tyhjä alkio ja se käsitellään median soittovaiheessa.

Ohjelmakoodin 1 *contents*-olion *baseurl*-merkkijono kertoo sovellukselle jokaisen tiedoston pääosoitteen, jossa tiedostot sijaitsevat. Tiedostojen muutoksien havaitsemiseen käytetään v*ersion*- ja *edit*-merkkijonoja. Merkkijonoja käytetään kun tiedostoa on muutettu, mutta sen osoite on edelleen sama.

Ohjelmakoodin 1 c*ontent*-olion *id*-jonon alkioiksi asetetaan tiedostojen sijaintikansioiden nimet, joita käytetään tiedostojen löytämiseen niiden latausvaiheessa. Tiedostojen tyypit, joita käytetään oikean toistomekanismin päättämiseen median toistovaiheessa, kerätään c*ontentType*-jonoon. Tiedostojen kansioiden versiot, jotka tarvittaessa helpottavat muuttuneiden tiedostojen löytämistä, asetetaan alkioiksi *version*-jonoon.

Ohjelmakoodin 1 *file*-olion ominaisuuksiin kuuluvat *parentId*-, *name*- ja *md5*-jonot. Tiedoston sijaintikansio määritetään *parentId*-jonon alkioiden avulla. Tiedostojen nimet asetetaan päätteineen *name*-jonoon ja niitä käytetään median latausvaiheessa. Tiedostojen muuttuminen tarkastetaan *md5*-jonon alkioiden avulla, joiksi asetetaan MD5-algoritmilla muodostetut tiedostokohtaiset merkkijonot.

Ohjelmakoodin 1 *Parser*-olion *xmlLocal*- ja *xmlExternal*-muuttujiin asetetaan ajax-pyynnöillä saadut älytelevision muistissa ja palvelimella olevien XML-tiedostojen informaatiot. *Parser*-olion muuttujiin sisältyy myös *newPlaylist*-olio. Uuden soittolistan muodostamiseen käytetyn newPlaylist-olion muuttujiin kuuluu *url*-, *type*- ja *duration*-jonot sekä *items*-kokonaisluku. Tiedostojen täydelliset osoitteet kerätään *url*-jonoon. Soittolistojen medioiden tyypit asetaan *type*-jonon alkioiksi. Medioiden kestot kerätään *duration*-jonoon. Älytelevision sisäisessä muistissa olevien, soittolistaan kuuluvien, tiedostojen määrä lasketaan *items*-kokonaisluvun avulla. Sovelluksessa *items*-kokonaisluku päivitetään kuitenkin vasta tiedostojen latausvaiheessa.

3.2.2 Ajax-pyynnöt

XML-tiedostojen pyynnöissä käytetään jQuery-kirjaston ajax-pyyntöjä. Ajax-pyyntöjen tyyppinä käytetään oletuksena olevaa GET-tyyppiä.

Ohjelmakoodi 2. Parser.GETLocalXML-metodi.

```
Parser.GETLocalXML = function() {
   Main.log("</br>Parser.GETLocalXML()");
   jQuery.ajax({
      url: "file://" + FileManager.xmlLocalURL,
      dataType: "xml",
      async: false,
      success: function(data) {
         Parser.xmlLocal = jQuery(data);
         Main.log("Local XML file parsed");
         Main.log("Local version: " + jQuery(data).find("display").attr("version"));
         Parser.GETExternalXML();
      },
      error: function(jqXHR, textStatus, errorThrown) {
         Main.log("Error while parsing Local XML file");
         Main.log("jqXHR: " + jqXHR);
         Main.log("textStatus: " + textStatus);
         Main.log("errorThrown: " + jqXHR.status + " (" + errorThrown + ")");
      }
   });
}
```

Ohjelmakoodissa 2 Ajax-pyyntö sisäisessä muistissa olevasta XML-tiedostosta lähetetään *GETLocalXML*-nimisestä metodista, joka on *Parser*-olion metodi. Pyynnössä määritetään datan osoite, tyyppi, lukemisen asynkronisuus ja datan hakemisen onnistuessa sekä epäonnistuessa tehtävät asiat. Datan osoitteen parametriksi asetetaan tiedoston sijainti älytelevision sisäisessä muistissa. Ajax-pyyntö käyttää oletuksena älykästä arvausta datan tyypin päättelyssä, mutta virheiden välttämiseksi määritetään se erikseen XML:ksi. Datan lukemisen

asynkronisuus asetetaan epätodeksi, koska datan saaminen sisäisessä muistissa sijaitsevasta XML-tiedostosta on välttämätöntä ennen ohjelmakoodissa etenemistä. Datan hakemisen onnistuessa asetetaan data *xmlLocal*-muuttujaan ja edetään ohjelmakoodissa *GETExternalXML*-metodiin. Datan hakemisen epäonnistuessa tulostetaan älytelevision ruudulle ajax-pyynnön virheen palauttamat parametrit.

Ohjelmakoodi 3. Parser.GETExternalXML-metodi.

```
Parser.GETExternalXML = function() {
Main.log("</br>FileManager.GETExternalXML()");
   jQuery.ajax({
      url: FileManager.xmlExternalURL,
      dataType: "xml",
      async: false,
      success: function(data) {
         Parser.xmlExternal = jQuery(data);
         Main.log("External XML file parsed");
         Main.log("External version: " + jQuery(data).find("display").attr("version"));
         FileManager.checkVersion();
      },
      error: function(jqXHR, textStatus, errorThrown) {
         Main.log("Error while parsing External XML file");
         Main.log("jqXHR: " + jqXHR);
         Main.log("textStatus: " + textStatus);
         Main.log("errorThrown: " + jqXHR.status + " (" + errorThrown + ")");
         FileManager.checkVersion();
      }
   });
}
```

Ohjelmakoodin 3 *GETExternalXML*-metodissa lähetetään ajax-pyyntö palvelimella sijaitsevasta XML-tiedostosta ja se on lähes identtinen *GETLocalXML*-metodin kanssa. Eroavuudet ovat datan osoitteessa ja datan hakemisen onnistuessa sekä epäonnistuessa tehtävissä asioissa. Datan osoitteeksi asetetaan palvelimella sijaitsevan XML-tiedoston HTTP-osoite.

Datan hakemisen onnistuessa asetetaan data *xmlExternal*-muuttujaan ja jatketaan ohjelmakoodissa metodiin, joka tarkistaa onko päivitystä XML-tiedostoon saatavilla. Datan hakemisen epäonnistuessa tulostetaan älytelevision ruudulle ajax-pyynnön virheen parametrit ja jatketaan samaan metodiin kuin datan hakemisen onnistuessa.

3.2.3 XML-tiedoston datan jäsentely

XML-tiedoston datan jäsentely suoritetaan jQuery-kirjaston avulla. Ajax-pyyntöjen suorittamisen jälkeen voidaan hakea jQuery-kirjaston avulla yksittäisiä tietoja muuttujista, joihin XML-tiedostojen datat asetetaan.

Datan jäsentely suoritetaan *parseXML*-nimisessä metodissa, joka on *Parser*-olion metodi. Aluksi metodissa tyhjennetään *presentation*-, *contents*-, *content*- ja *file*-muuttujat, koska sovelluksen käynnistymisen lisäksi metodi suoritetaan myös päivityksen yhteydessä. Muuttujien tyhjentämisen jälkeen aloitetaan datan jäsentäminen tyhjennettyihin muuttujiin (liite 1).

XML-elementtien jäsentämiset toteutetaan pääosin samalla tavalla jokaisen muuttujan kohdalla. Ajax-pyynnöllä haetusta datasta etsitään elementtejä tietyllä nimellä. Elementtien etsintä palauttaa löydetyt elementit jonossa, joka asetetaan avustavaan muuttujaan. Jonon alkiot käydään läpi for-silmukassa ja niistä haetaan jQuery-kirjaston avulla etsityn elementin muuttujia vastaavat attribuutit, jotka asetetaan etsityn elementin muuttujiin (liite 1).

Eroavaisuuksia on *contents*-olion muuttujien arvojen hakemisessa, koska XML-skeeman mukaan *contents*-elementtejä voi olla vain yksi ja *contets*-olion muuttujina on vain merkkijonoja. Löydettävien elementtien ollessa vain yksi voidaan jättää for-silmukan käyttö pois ja määrittää käytettäväksi jonon ensimmäistä alkiota. Toisena eroavaisuutena on *file*-olion *parentId*-jono, jonka alkiot asetetaan jo *content*-olion attribuuttien kanssa. Attribuutteja haettaessa *content*-oliosta etsitään samalla sen alle kuuluvien tiedostojen määrä ja asetetaan *content*-olion *id*-jonon alkio löydettyjen tiedostojen määrän verran *file*-olion *parentId*-jonoon (liite 1).

3.2.4 Muuttuneiden listan muodostus

Muuttuneiden listaan kerätään tiedostot, jotka kuuluvat toistettavaan presentaatioon, mutta niistä on tullut päivityksen yhteydessä uusi versio saataville. Muuttuneen tiedoston nimi on tällöin sama, mutta sen versio-numerointi on muuttunut. Muuttuneiden listan muodostus tehdään *Parser*-olion *createChangedList*-metodissa. Metodi suoritetaan, mikäli XML-tiedostosta on julkaistu uusi versio.

Metodissa etsitään aluksi XML-tiedostojen datat sisältävistä muuttujista *contents*-elementit ja niiden attribuuteista verrataan *edit*-muuttujien arvoja. Arvojen ollessa samat oletetaan presentaatiossa käytettävien tiedostojen versioiden olevan samat. Arvojen ollessa erisuuruiset jatketaan ohjelmakoodissa etsimään XML-tiedostoista *content*-elementtejä (liite 2).

Etsityt *content*-elementit asetetaan uusiin muuttujiin ja luodaan *contLAttr*- ja *contEAttr*-oliot, joiden muuttujiksi alustetaan *id*- ja *version*-jonot. Olioiden luomisen ja muuttujien alustamisen jälkeen käytetään for-silmukkaa, jonka sisällä haetaan jonojen alkioiksi *content*-elementtien *id*- ja *version*-attribuutit. Jonojen täyttäminen toteutetaan samalla tavalla sisäisessä muistissa olevalle ja palvelimella sijaitsevalle XML-tiedostolle (liite 2).

Jonojen täyttämisen jälkeen suoritetaan for-silmukka sisäisessä muistissa olevan XML-tiedostoston *content*-elementtien lukumäärän verran. Silmukassa luodaan muuttuja, jonka arvoksi asetetaan *contLAttr*-olion *id*-jonon alkion indeksinumero *contEAttr*-olion *id*-jonossa. Muuttujan arvoksi saadaan -1, mikäli alkiota ei löydy *contEAttr*-olion *id*-jonosta. Muuttujan arvon ollessa suurempi kuin -1 verrataan käytetyn *contLAttr*-olion *id*-jonon alkiota vastaavaa *version*-jonon alkiota ja löydetyn *contEAttr*-olion *id*-jonon alkiota vastaavaa *version*-jonon alkiota. Verrattujen alkioiden ollessa erisuuruisia etsitään sisäisessä muistissa olevan XML-tiedoston datasta käsiteltävän *content*-elementin sisältämät *file*-elementit, joista etsitään *name*-attribuutit. Löydetyt attribuutit lisätään muuttuneiden tiedostojen listaan. Mikäli silmukan suorittamisen jälkeen

muuttuneiden lista on tyhjä, päätellään kaikkien presentaatiossa käytettävien tiedostojen olevan muuttumattomia (liite 2).

3.2.5 Latauslistan muodostus

Latauslistaan kerätään kaikki presentaatioon kuuluvat tiedostot, jotka ladataan latausvaiheessa älytelevision sisäiseen muistiin. Sisäisessä muistissa olevat tiedostot siivotaan pois latauslistasta ennen median lataamisen aloittamista.

Ohjelmakoodi 4. Parser.createDownloadList-metodi.

```
Parser.createDownloadList = function() {
   Main.log("</br>Parser.createDownloadList()");
   FileManager.downloadList = {baseURL:[], fileName:[], fetched:0};

   for(var i=0; i<presentation.contentId.length; i++) {
      for(var j=0; j<content.id.length; j++) {
         if(presentation.contentId[i] == content.id[j]) {
            for(var k=0; k<file.name.length; k++) {
               if(file.parentId[k] == content.id[j]) {
                  FileManager.downloadList.baseURL.push(contents.baseurl +
                  content.id[j] + "/");
                  FileManager.downloadList.fileName.push(file.name[k]);
               }
            }
         }
      }
   }

   Main.log("Download list created");
   if(!Main.xmlUpdate)
      Parser.createPlaylist();
   else
      Parser.createNewPlaylist();
}
```

Ohjelmakoodin 4 *createDownloadList*-metodin alussa tyhjennetään *download-List*-olion *baseURL*- ja *fileName*-jonot sekä asetetaan *fetched*-kokonaislukun arvoksi 0. Muuttujien alustamisen jälkeen suoritetaan for-silmukkaa *presentation*-olion *contentId*-jonon alkioiden lukumäärän verran. Silmukassa suoritetaan toinen for-silmukka, jota käydään läpi *content*-olion *id*-jonon alkioiden lukumäärän verran.

Toisessa silmukassa verrataan *presentation*-olion *contentId*-jonon alkiota *content*-olion *id*-jonon alkioon. Mikäli alkiot ovat samansuuruiset, suoritetaan kolmas for-silmukka *file*-olion *name*-jonon pituuden verran.

Kolmannen silmukan sisällä verrataan *file*-olion *parentId*-jonon alkiota *content*-olion *id*-jonon alkioon. Mikäli alkiot ovat samansuuruiset, lisätään *downloadList*-olion *baseURL*- ja *fileName*-jonoihin uudet alkiot. Olion *baseURL*-jonoon lisätään *contents*-olion *baseurl*-merkkijono sekä *content*-olion *id*-jonon alkio ja *fileName*-jonoon lisätään *file*-olion *name*-jonon alkio.

Silmukoiden suorittamisen jälkeen siirrytään joko *createPlaylist*-metodiin tai *createNewPlaylist*-metodiin. Metodin valinta riippuu siitä, suoritetaanko metodi käynnistyksen yhteydessä vai päivityksen yhteydessä.

3.2.6 Soittolistojen muodostus

Soittolistaan asetetaan presentaation medioiden toistolle olennaiset tiedot XML-tiedostossa määrätyssä toistojärjestyksessä. Soittolistoja on kaksi erilaista. Ensimmäistä soittolistaa käytetään median toistamiseen. Toiseen soittolistaan muodostetaan päivityksen yhteydessä uusi soittolista. Sovelluksen käynnistymisen yhteydessä suoritetaan *createPlaylist*-metodi, joka muodostaa ensimmäisen soittolistan. Sovelluksen päivityksen yhteydessä suoritetaan *createNewPlaylist*-metodi ja se muodostaa toisen soittolistan, jolla korvataan ensimmäinen soittolista, uuden median ensimmäisen tiedoston lataamisen jälkeen.

Ohjelmakoodi 5. Parser.createPlaylist-metodi.

```
Parser.createPlaylist = function() {
   Main.log("</br>Parser.createPlaylist()");

   for(var i=0; i<presentation.contentId.length; i++) {
      for(var j=0; j<content.id.length; j++) {
         if(presentation.contentId[i] == content.id[j]) {
            for(var k=0; k<file.name.length; k++) {
               if(file.parentId[k] == content.id[j]) {
                  MediaPlayer.playlist.url.push(FileManager.mediaDir + file.name[k]);
                  MediaPlayer.playlist.type.push(content.contentType[j]);
                  MediaPlayer.playlist.duration.push(presentation.duration[i]);
               }
            }
         }
      }
   }
   Main.log("Playlist created");
   FileManager.checkMediaFiles();
}
```

Sovelluksen käynnistyessä suoritetaan ohjelmakoodin 5 *Parser*-olion *createPlaylist*-metodi, jonka tarkoituksena on täyttää median toistamiseen käytettävä soittolista presentaatioon kuuluvien medioiden tietojen kanssa. Aluksi metodissa suoritetaan for-silmukka *presentation*-olion *contentId*-jonon alkioiden lukumäärän verran. Silmukan sisällä suoritetaan toinen for-silmukka *content*-olion *id*-jonon alkioiden lukumäärän verran.

Toisen silmukan sisällä verrataan *presentation*-olion *contentId*-jonon alkiota *content*-olion *id*-jonon alkioon. Mikäli alkiot ovat samansuuruiset, suoritetaan kolmas for-silmukka *file*-olion *name*-jonon pituuden verran.

Kolmannessa silmukassa verrataan *file*-olion *parentId*-jonon alkiota *content*-olion *id*-jonon alkioon. Alkioiden ollessa samansuuruiset, asetaan *playlist*-olion *url*-jonoon sisäisessä muistissa sijaitsevan mediakansion osoite sekä *file*-olion

name-jonon alkio. Asetettua alkiota käytetään median toistovaiheessa tiedoston osoitteena. Osoitteen asettamisen jälkeen *playlist*-olion *type*-jonoon lisätään *content*-olion *contetType*-jonon alkio, jota käytetään median oikean toistomekanismin päättämiseen. Median tyypin lisäämisen jälkeen lisätään *playlist*-olion *duration*-jonoon *presentation*-olion *duration*-jonon alkio, jolla määritetään median toistamisen kesto. Median toistamisen kestoa käytetään vain toistettaessa kuvia.

Silmukoiden suorittamisen jälkeen siirrytään *checkMediaFiles*-metodiin. Metodissa tarkastetaan älytelevision sisäisessä muistissa olevat mediatiedostot ja presentaatioon kuulumattomat tiedostot poistetaan.

Ohjelmakoodi 6. Parser.createNewPlaylist-metodi.

```
Parser.createNewPlaylist = function() {
   Main.log("</br>Parser.createNewPlaylist()");
   this.newPlaylist = {url:[], type:[], duration:[], items:0};

   for(var i=0; i<presentation.contentId.length; i++) {
      for(var j=0; j<content.id.length; j++) {
         if(presentation.contentId[i] == content.id[j]) {
            for(var k=0; k<file.name.length; k++) {
               if(file.parentId[k] == content.id[j]) {
                  this.newPlaylist.url.push(FileManager.mediaDir + file.name[k]);
                  this.newPlaylist.type.push(content.contentType[j]);
                  this.newPlaylist.duration.push(presentation.duration[i]);
               }
            }
         }
      }
   }
   Main.log("New playlist created");
   FileManager.checkMediaFiles();
}
```

Uuden toistolistan muodostaminen suoritetaan Ohjelmakoodin 6 *Parser*-olion *createNewPlaylist*-metodissa, joka on ohjelmakoodilta lähes identtinen *createPlaylist*-metodin kanssa. Erovaisuuksina on ohjelmakoodin alussa suoritettava *newPlaylist*-olion *url*-, *type*-, *duration*- ja *items*-muuttujien alustaminen sekä *newPlaylist*-olion käyttäminen *playlist*-olion sijaan.

3.2.7 Testaus ja virheiden etsiminen

Varsinaisen kehitysympäristön puuttuessa käytetään *Main*-olion *log*-metodia sovelluksen koodin testaukseen ja virheiden etsimiseen. Metodi tulostaa älytelevision näytölle HTML-koodin kautta erinäisiä viestejä, jotka voivat merkitä ohjelmakoodin etenemistä tai siinä tapahtuneita virheitä.

Ohjelmakoodi 7. Main.log-metodi.

```
Main.log = function(msg) {
   if(this.debug) {
      this.debugObj.innerHTML += "</br><div style='color:#FFFFFF;" +
      "background-color:#000000; display:inline-block'>" + msg + "</div>";

      if(this.debugObj.childNodes.length > 40) {
         this.debugObj.removeChild(this.debugObj.childNodes[1]);
         this.debugObj.removeChild(this.debugObj.childNodes[0]);
      }
   }
}
```

Ohjelmakoodin 7 *Main*-olion *log*-metodissa tarkastetaan ensimmäisenä *debug*-muuttuja, jonka arvo määrittää tulostetaanko älytelevision näyttöruudulle ohjelmakoodin etenemistä kuvaavia tekstejä. Mikäli *debug*-muuttujan arvo on tosi, lisätään HTML-koodissa olevaan *debugObj*-elementtiin HTML-koodin br-merkki tarkoittamaan rivin lopetusta ja HTML-koodin div-elementin sisällä metodiin lähetetty *msg*-merkkijono. HTML-koodissa oleva *debugObj*-elementti on HTML-koodin div-elementti.

HTML-koodissa olevaan *debugObj*-elementtiin lisäämisen jälkeen verrataan *debugObj*-elementin sisältämien elementtien määrää ja jos määrä ylittää arvon 40, poistetaan *debugObj*-elementin sisältä kaksi ensimmäistä elementtiä. Elementtien poistaminen suoritetaan ohjelmakoodin etenemisen sekä virheiden etsimisen selkeyttämiseksi. Elementtien poistaminen ehkäisee samalla älytelevision suoritustehon laskemisen suuren div-elementtien määrän takia.

4 YHTEENVETO

Opinnäytetyössä kuvattiin Samsung Smart Signage Platform -järjestelmässä toimivan sovelluksen yhdistäminen Qem Software Oy:n Digital Signage -järjestelmään XML-merkintäkielen avulla. Työssä paneuduttiin järjestelmien yhdistämisessä käytettyyn ohjelmakoodiin käsittelemällä sen toimintaa.

Työn toteutuksesta saatiin toimiva kokonaisuus, jonka ratkaisua voidaan soveltaa vastaavanlaisiin sovelluksiin. Järjestelmien yhdistämisen toteutuksesta tehtiin helppolukuinen ja rakenteeltaan looginen, jotta sitä voidaan tulevaisuudessa jatkaa ja laajentaa. Toteutusta voi laajentaa esimerkiksi suurentamalla XML-tiedoston jäsentämiseen käytettyjen elementtien määrää. Laajennettaessa jäsennettävien elementtien määrää elementeille tulee kuitenkin olla käyttötarkoitus sovelluksessa, jotta vältetään turhan ohjelmankoodin lisääminen sovellukseen.

LÄHTEET

[1] W3Schools. 2014. Introduction to XML. Viitattu 13.11.2014 http://www.w3schools.com/xml/xml_whatis.asp.

[2] The XML FAQ. 2006. Why should I use XML?. Viitattu 13.11.2014 http://xml.silmaril.ie/whyxml.html.

[3] The UK Web Design Company. 2014. XML: The Well-formed Document. Viitattu 14.11.2014 http://www.theukwebdesigncompany.com/articles/xml-well-formed.php.

[4] W3Schools. 2014. XML Document Types. Viitattu 14.11.2014 http://www.w3schools.com/xml/xml_doctypes.asp.

[5] Wrox. 2000. Beginning XML - Chapter 2: Well-formed XML. Viitattu 14.11.2014 http://www.codeproject.com/Articles/845/Beginning-XML-Chapter-Well-Formed-XML.

[6] W3Schools. 2014. XML Schema. Viitattu 15.11.2014 http://www.w3schools.com/xml/xml_schema.asp.

[7] The jQuery Foundation. 2014. jQuery: Write less, do more. Viitattu 18.11.2014 http://jquery.com/.

[8] Sheo Narayan. 2011. What is jQuery and How to Start using jQuery?. Viitattu 18.11.2014 http://www.codeproject.com/Articles/157446/What-is-jQuery-and-How-to-Start-using-jQuery.

[9] Dykes, L. ja Ullman C. 2007. Beginning Ajax. Indianapolis: Wiley Publishing, Inc.

[10] BroadSign. 2014. What is Digital Signage?. Viitattu 20.11.2014 http://broadsign.com/what-is-digital-signage/.

[11] Qem Software. 2014. Qem Digital Sigange. Viitattu 21.11.2014 http://qem.fi/.

[12] Signagelive. 2014. What is smart signage?. Viitattu 22.11.2014 http://signagelive.com/smartlfd/.

[13] Qem Software Oy CTO Timo Koski. Haastattelu 24.11.2014.

Ohjelmakoodi XML-tiedoston datan jäsentämiseen

```
Parser.parseXML = function(xml) {
   Main.log("</br>Parser.parseXML()");

   //Empty attribute objects
   presentation = {contentId:[], duration:[]};
   contents = {baseurl:"", version:"", edit:""};
   content = {id:[], contentType:[], version:[]};
   file = {parentId:[], name:[], md5:[]};

   //Parse presentation attributes
   var pres = xml.find("presentation");
   for(var i=0; i<pres.length; i++) {
      presentation.contentId[i] = jQuery(pres[i]).attr("contentId");
      presentation.duration[i] = jQuery(pres[i]).attr("duration");
   }
   //Parse contents attributes
   var conts = xml.find("contents");
   contents.baseurl = jQuery(conts[0]).attr("baseurl");
   contents.version = jQuery(conts[0]).attr("version");
   contents.edit = jQuery(conts[0]).attr("edit");

   //Parse content attributes
   var cont = xml.find("content");
   for(var i=0; i<cont.length; i++) {
      content.id[i] = jQuery(cont[i]).attr("id");
      content.contentType[i] = jQuery(cont[i]).attr("contentType");
      content.version[i] = jQuery(cont[i]).attr("version");

      //Parse file parents
      var fil = jQuery(cont[i]).find("file");
      for(var j=0; j<fil.length; j++) {
         file.parentId.push(content.id[i]);
      }
```

```
    }

    //Parse file attributes
    var fil = xml.find("file");
    for(var i=0; i<fil.length; i++) {
       file.name[i] = jQuery(fil[i]).attr("name");
       file.md5[i] = jQuery(fil[i]).attr("md5");
    }

    Main.log("XML Parsing complete");
    Parser.createDownloadList();
}
```

Ohjelmakoodi muuttuneiden listan muodostamiseen

```
Parser.createChangedList = function() {
   Main.log("Create list of changed media files if there is any:");
   FileManager.changedList = [];
   var contsL = this.xmlLocal.find("contents");
   var contsE = this.xmlExternal.find("contents");

   if(jQuery(contsL[0]).attr("edit") != jQuery(contsE[0]).attr("edit")) {
      //Get localXML content attributes
      var contL = this.xmlLocal.find("content");
      var contLAttr = {id:[], version:[]};
      for(var i=0; i<contL.length; i++) {
         contLAttr.id[i] = jQuery(contL[i]).attr("id");
         contLAttr.version[i] = jQuery(contL[i]).attr("version");
      }
      //Get externalXML content attributes
      var contE = this.xmlExternal.find("content");
      var contEAttr = {id:[], version:[]};
      for(var i=0; i<contE.length; i++) {
         contEAttr.id[i] = jQuery(contE[i]).attr("id");
         contEAttr.version[i] = jQuery(contE[i]).attr("version");
      }

      for(var i=0; i<contL.length; i++) {
         var contIndex = contEAttr.id.indexOf(contLAttr.id[i]);
         if(contIndex > -1 && contLAttr.version[i] != contEAttr.version[contIndex]) {
            Main.log("Changed file: " + contLAttr.id[i]);
            var changedFile = jQuery(contL[i]).find("file");
            for(var j=0; j<changedFile.length; j++) {
              FileManager.changedList.push(jQuery(changedFile[j]).attr("name"));
            }
         }
      }
      if(FileManager.changedList.length == 0)
```

```
            Main.log("No file changes");
        }
        else
            Main.log("No content changes");
    }
```